PORTAL
USPTO

Subscribe (Full Service) Register (Limited Service, Free) Login

Search: ◉ The ACM Digital Library ○ The Guide

verify + assembly + "run time" + manifest + public + key SEARCH

THE ACM DIGITAL LIBRARY

Feedback Report a problem Satisfaction survey

Terms used **verify assembly run time manifest public key**

Found **29,217** of **177,263**

Sort results by: relevance
Display results: expanded form

Save results to a Binder
Search Tips
☐ Open results in a new window

Try an Advanced Search
Try this search in The ACM Guide

Results 1 - 20 of 200 Result page: **1** 2 3 4 5 6 7 8 9 10 next

Best 200 shown

Relevance scale

1 Type-Safe linking with recursive DLLs and shared libraries
Dominic Duggan
November 2002 **ACM Transactions on Programming Languages and Systems (TOPLAS)**, Volume 24 Issue 6
Publisher: ACM Press
Full text available: pdf(658.62 KB) Additional Information: full citation, abstract, references, citings, index terms

Component-based programming is an increasingly prevalent theme in software development, motivating the need for expressive and safe module interconnection languages. Dynamic linking is an important requirement for module interconnection languages, as exemplified by dynamic link libraries (DLLs) and class loaders in operating systems and Java, respectively. A semantics is given for a type-safe module interconnection language that supports shared libraries and dynamic linking, as well as circular ...

Keywords: Dynamic Linking, Module Interconnection Languages, Recursive Modules, Shared Libraries

2 Special issue: AI in engineering
D. Sriram, R. Joobbani
April 1985 **ACM SIGART Bulletin**, Issue 92
Publisher: ACM Press
Full text available: pdf(8.79 MB) Additional Information: full citation, abstract

The papers in this special issue were compiled from responses to the announcement in the July 1984 issue of the SIGART newsletter and notices posted over the ARPAnet. The interest being shown in this area is reflected in the sixty papers received from over six countries. About half the papers were received over the computer network.

3 Proof linking: modular verification of mobile programs in the presence of lazy, dynamic linking
Philip W. L. Fong, Robert D. Cameron
October 2000 **ACM Transactions on Software Engineering and Methodology (TOSEM)**, Volume 9 Issue 4
Publisher: ACM Press
Full text available: pdf(233.60 KB) Additional Information: full citation, abstract, references, citings, index

terms, review

Although mobile code systems typically employ link-time code verifiers to protect host computers from potentially malicious code, implementation flaws in the verifiers may still leave the host system vulnerable to attack. Compounding the inherent complexity of the verification algorithms themselves, the need to support lazy, dynamic linking in mobile code systems typically leads to architectures that exhibit strong interdependencies between the loader, the verifier, and the linker. To simp ...

Keywords: Java, correctness conditions, dynamic linking, mobile code, modularity, proof linking, safety, verification protocol, virtual machine architecture

4 GPGPU: general purpose computation on graphics hardware
David Luebke, Mark Harris, Jens Krüger, Tim Purcell, Naga Govindaraju, Ian Buck, Cliff Woolley, Aaron Lefohn
August 2004 **Proceedings of the conference on SIGGRAPH 2004 course notes GRAPH '04**
Publisher: ACM Press
Full text available: pdf(63.03 MB) Additional Information: full citation, abstract

The graphics processor (GPU) on today's commodity video cards has evolved into an extremely powerful and flexible processor. The latest graphics architectures provide tremendous memory bandwidth and computational horsepower, with fully programmable vertex and pixel processing units that support vector operations up to full IEEE floating point precision. High level languages have emerged for graphics hardware, making this computational power accessible. Architecturally, GPUs are highly parallel s ...

5 Fast detection of communication patterns in distributed executions
Thomas Kunz, Michiel F. H. Seuren
November 1997 **Proceedings of the 1997 conference of the Centre for Advanced Studies on Collaborative research**
Publisher: IBM Press
Full text available: pdf(4.21 MB) Additional Information: full citation, abstract, references, index terms

Understanding distributed applications is a tedious and difficult task. Visualizations based on process-time diagrams are often used to obtain a better understanding of the execution of the application. The visualization tool we use is Poet, an event tracer developed at the University of Waterloo. However, these diagrams are often very complex and do not provide the user with the desired overview of the application. In our experience, such tools display repeated occurrences of non-trivial commun ...

6 A structural view of the Cedar programming environment
Daniel C. Swinehart, Polle T. Zellweger, Richard J. Beach, Robert B. Hagmann
August 1986 **ACM Transactions on Programming Languages and Systems (TOPLAS)**, Volume 8 Issue 4
Publisher: ACM Press
Full text available: pdf(6.32 MB) Additional Information: full citation, abstract, references, citings, index terms

This paper presents an overview of the Cedar programming environment, focusing on its overall structure—that is, the major components of Cedar and the way they are organized. Cedar supports the development of programs written in a single programming language, also called Cedar. Its primary purpose is to increase the productivity of programmers whose activities include experimental programming and the development of prototype software systems for a high-performance personal computer. T ...

7 Attribute grammar paradigms—a high-level methodology in language implementation

Jukka Paakki
June 1995 **ACM Computing Surveys (CSUR)**, Volume 27 Issue 2
Publisher: ACM Press
Full text available: pdf(5.15 MB) Additional Information: full citation, abstract, references, citings, index terms, review

Attribute grammars are a formalism for specifying programming languages. They have been applied to a great number of systems automatically producing language implementations from their specifications. The systems and their specification languages can be evaluated and classified according to their level of application support, linguistic characteristics, and degree of automation.A survey of attribute grammar-based specification languages is given. The modern advanced specification ...

Keywords: attribute grammars, blocks, classes, compiler writing systems, functional dependencies, incomplete data, incrementality, inheritance, language processing, language processor generators, lazy evaluation, logical variables, objects, parallelism, processes, programming paradigms, semantic functions, symbol tables, unification

8 High level programming for distributed computing
Jerome A. Feldman
June 1979 **Communications of the ACM**, Volume 22 Issue 6
Publisher: ACM Press
Full text available: pdf(1.78 MB) Additional Information: full citation, abstract, references, citings

Programming for distributed and other loosely coupled systems is a problem of growing interest. This paper describes an approach to distributed computing at the level of general purpose programming languages. Based on primitive notions of module, message, and transaction key, the methodology is shown to be independent of particular languages and machines. It appears to be useful for programming a wide range of tasks. This is part of an ambitious program of development in advanced programmin ...

Keywords: assertions, distributed computing, messages, modules

9 Type-based hot swapping of running modules (extended abstract)
Dominic Duggan
October 2001 **ACM SIGPLAN Notices , Proceedings of the sixth ACM SIGPLAN international conference on Functional programming ICFP '01**, Volume 36 Issue 10
Publisher: ACM Press
Full text available: pdf(150.34 KB) Additional Information: full citation, abstract, references, citings, index terms

While dynamic linking has become an integral part of the run-time execution of modem programming languages, there is increasing recognition of the need for support for hot swapping of running modules, particularly in long-lived server applications. The interesting challenge for such a facility is to allow the new module to change the types exported by the original module, while preserving type safety. This paper describes a type-based approach to hot swapping running modules. The approach is bas ...

Keywords: dynamic typing, hot swapping, module interconnection languages, shared libraries

10 Units: cool modules for HOT languages
Matthew Flatt, Matthias Felleisen

May 1998 **ACM SIGPLAN Notices , Proceedings of the ACM SIGPLAN 1998 conference on Programming language design and implementation PLDI '98**, Volume 33 Issue 5
Publisher: ACM Press
Full text available: pdf(1.54 MB) Additional Information: full citation, abstract, references, citings, index terms

A module system ought to enable *assembly-line programming* using separate compilation and an expressive linking language. Separate compilation allows programmers to develop parts of a program independently. A linking language gives programmers precise control over the assembly of parts into a whole. This paper presents models of *program units*, MzScheme's module language for assembly-line programming. Units support separate compilation, independent module reuse, cyclic dependencies, ...

11 Task-structure analysis for knowledge modeling
B. Chandrasekaran, Todd R. Johnson, Jack W. Smith
September 1992 **Communications of the ACM**, Volume 35 Issue 9
Publisher: ACM Press
Full text available: pdf(2.77 MB) Additional Information: full citation, references, citings, index terms

Keywords: analysis, modeling

12 Software safety: why, what, and how
Nancy G. Leveson
June 1986 **ACM Computing Surveys (CSUR)**, Volume 18 Issue 2
Publisher: ACM Press
Full text available: pdf(4.18 MB) Additional Information: full citation, abstract, references, citings, index terms, review

Software safety issues become important when computers are used to control real-time, safety-critical processes. This survey attempts to explain why there is a problem, what the problem is, and what is known about how to solve it. Since this is a relatively new software research area, emphasis is placed on delineating the outstanding issues and research topics.

13 A new model of security for distributed systems
Wm A. Wulf, Chenxi Wang, Darrell Kienzle
September 1996 **Proceedings of the 1996 workshop on New security paradigms**
Publisher: ACM Press
Full text available: pdf(1.10 MB) Additional Information: full citation, references, citings, index terms

14 The complexity of logic-based abduction
Thomas Eiter, Georg Gottlob
January 1995 **Journal of the ACM (JACM)**, Volume 42 Issue 1
Publisher: ACM Press
Full text available: pdf(3.02 MB) Additional Information: full citation, abstract, references, citings, index terms, review

Abduction is an important form of nonmonotonic reasoning allowing one to find explanations for certain symptoms or manifestations. When the application domain is described by a logical theory, we speak about logic-based abduction. Candidates for abductive explanations are usually subjected to minimality criteria such as subset-minimality, minimal cardinality, minimal weight, or minimality under prioritization of individual hypotheses. This paper presents a comprehensive com ...

Keywords: abduction, complexity analysis, diagnosis, propositional logic, reasoning

15 Early experience with Mesa
Charles M. Geschke, James H. Morris, Edwin H. Satterthwaite
August 1977 **Communications of the ACM**, Volume 20 Issue 8
Publisher: ACM Press
Full text available: pdf(1.46 MB) Additional Information: full citation, abstract, references, citings

The experiences of Mesa's first users—primarily its implementers—are discussed, and some implications for Mesa and similar programming languages are suggested. The specific topics addressed are: module structure and its use in defining abstractions, data-structuring facilities in Mesa, an equivalence algorithm for types and type coercions, the benefits of the type system and why it is breached occasionally, and the difficulty of making the treatment of variant records safe.
Keywords: data structures, modules, programming languages, systems programming, types

16 Automated and certified conformance to responsiveness policies
Joseph C. Vanderwaart, Karl Crary
January 2005 **Proceedings of the 2005 ACM SIGPLAN international workshop on Types in languages design and implementation**
Publisher: ACM Press
Full text available: pdf(615.49 KB) Additional Information: full citation, abstract, references, index terms

Certified code systems protect computers from faulty or malicious code by requiring untrusted software to be accompanied by checkable evidence of its safety. This paper presents a certified code solution to a problem in grid computing, namely, controlling the CPU usage of untrusted programs. Specifically, we propose to endow the runtime system supervising local execution of grid programs with a trusted "yield" operation, and require the untrusted code to execute this operation with at least a ce ...

Keywords: certified code, grid computing, typed assembly language

17 Projectors: advanced graphics and vision techniques
Ramesh Raskar
August 2004 **Proceedings of the conference on SIGGRAPH 2004 course notes GRAPH '04**
Publisher: ACM Press
Full text available: pdf(6.53 MB) Additional Information: full citation

18 A certifying compiler for Java
Christopher Colby, Peter Lee, George C. Necula, Fred Blau, Mark Plesko, Kenneth Cline
May 2000 **ACM SIGPLAN Notices , Proceedings of the ACM SIGPLAN 2000 conference on Programming language design and implementation PLDI '00**, Volume 35 Issue 5
Publisher: ACM Press
Full text available: pdf(792.48 KB) Additional Information: full citation, abstract, references, citings, index terms

This paper presents the initial results of a project to determine if the techniques of proof-carrying code and certifying compilers can be applied to programming languages of realistic size and complexity. The experiment shows that: (1) it is possible to implement a certifying native-code compiler for a large subset of the Java programming language; (2) the compiler is freely able to apply many standard local and global optimizations; and (3) the PCC bina ...

19 Frontmatter (TOC, Letter from the chair, Letter from the editor, Letters to the editor, ACM policy and procedures on plagiarism, PASTE abstracts, Calendar of future events, Workshop and conference information)
ACM SIGSOFT Software Engineering Notes staff
January 2006 **ACM SIGSOFT Software Engineering Notes**, Volume 31 Issue 1
Publisher: ACM Press
Full text available: pdf(1.82 MB) Additional Information: full citation, index terms

20 Automatically proving the correctness of compiler optimizations
Sorin Lerner, Todd Millstein, Craig Chambers
May 2003 **ACM SIGPLAN Notices , Proceedings of the ACM SIGPLAN 2003 conference on Programming language design and implementation PLDI '03**, Volume 38 Issue 5
Publisher: ACM Press
Full text available: pdf(285.83 KB) Additional Information: full citation, abstract, references, citings, index terms

We describe a technique for automatically proving compiler optimizations *sound*, meaning that their transformations are always semantics-preserving. We first present a domain-specific language, called Cobalt, for implementing optimizations as guarded rewrite rules. Cobalt optimizations operate over a C-like intermediate representation including unstructured control flow, pointers to local variables and dynamically allocated memory, and recursive procedures. Then we describe a technique for ...

Keywords: automated correctness proofs, compiler optimization

Results 1 - 20 of 200 Result page: **1** 2 3 4 5 6 7 8 9 10 next

WEST Search History

Hide Items | Restore | Clear | Cancel

DATE: Tuesday, May 30, 2006

Hide?	Set Name	Query	Hit Count
		DB=PGPB,USPT,USOC,EPAB,JPAB,DWPI,TDBD; PLUR=YES; OP=OR	
☐	L36	L35 and (manifest or ((meta adj adta or (file near2 structure))))	48
☐	L35	L34 and (public adj key and private adj key and signature and Hash$7)	360
☐	L34	L33 and ((software or application or assembly or code) same (authet$7 or verif$7))	889
☐	L33	713/187,176.ccls.	2017
☐	L32	717/167,178.ccls. and (public adj key)	25
☐	L31	717/$.ccls. and (manifest and public adj key)	33
☐	L30	(manifest and public adj key).clm.	24
☐	L29	((PARTHASARATHY near2 SRIVATSAN) or (PRATSCHNER near2 STEVEN) or (SINCLAIR near2 CRAIG)) and (manifest and public adj key)	10
☐	L28	L25 and (public same private same key) and (hash$7 near2 public adj key)	12
☐	L27	L25 and (public same private same key)	201
☐	L26	L25 and (public same private)	209
☐	L25	(380/30 |713/150 |713/155 |713/156 |713/170 |713/182 |713/189).ccls. and ((verify$7 or authenticat$7) near3 (software or code or asembly or application) same hash$7)	284
☐	L24	(380/30 |713/150 |713/155 |713/156 |713/170 |713/182 |713/189).ccls. and ((verify$7 or authgenticat$7) mnear3 (software or code or asembly or application) same hash$7)	2936
☐	L23	(authenticatinf near3 (software or code or asembly or DLL) near5 hash$7 same public same private)	0
☐	L22	(public adj key and private adj key and hash and manifest and referencing).clm.	0
☐	L21	(public adj key and private adj key and hash and manifest).clm.	8
☐	L20	(secure adj name and public adj key and private adj key and hash and manifest).clm.	0
☐	L19	709/$.ccls. and (secure adj name and public adj key and private adj key and hash and manifest)	0
☐	L18	715/$.ccls. and (secure adj name and public adj key and private adj key and hash and manifest)	0
☐	L17	709/$.ccls. and (secure adj name and public adj key and private adj key and hash and manifest)	0
☐	L16	380/$.ccls. and (secure adj name and public adj key and private adj key and hash and manifest)	0
☐	L15	713/$.ccls. and (secure adj name and public adj key and private adj key and hash and manifest)	2
☐	L14	726/$.ccls. and (secure adj name and public adj key and private adj key and hash and manifest)	0
☐	L12	(726/2).ccls. and (assembly same hash same public same private)	0
☐	L11	Parthasarathy.in. and manifest	21
☐	L10	(L5 or L2 or L3) and (public adj key near4 (file or directory))	66
☐	L9	(L5 or L2 or L3) and (public adj key near4 file or directory)	376

☐	L8	(L5 or L2 or L3) and (manifest same public adj key)	2
☐	L7	(L5 or L2 or L3) and (manifest same public adkj key)	1573
☐	L6	(L5 or L2 or L3) and manifest	23
☐	L5	(6754661 5892904 5958051 6108420 6425011 6901512 5790667 5892828 6069954 6249867 5818936 5943423 6028938 6078909 6202157 6378072 6470450 6499110 6694434 6804777 6317700 6202159 5343527 5802557 6108788 6131162 6192131 6256734 6298153 6367009 6367012 6381324 6438690 6715073 6789194 6198053 6192474 5805911 5896321 6134597 6154747 4624480 6243811 6810389 4924515 4853697 5180259 5294121 5418854 5497421).pn.	102
☐	L4	L3 (6754661 5892904 5958051 6108420 6425011 6901512 5790667 5892828 6069954 6249867 5818936 5943423 6028938 6078909 6202157 6378072 6470450 6499110 6694434 6804777 6317700 6202159 5343527 5802557 6108788 6131162 6192131 6256734 6298153 6367009 6367012 6381324 6438690 6715073 6789194 6198053 6192474 5805911 5896321 6134597 6154747 4624480 6243811 6810389 4924515 4853697 5180259 5294121 5418854 5497421).pn.	105
☐	L3	(713/200).ccls.	3
☐	L2	(713/189 |713/182).ccls.	1815
☐	L1	(6754661 5892904 5958051 6108420 6425011 6901512 5790667 5892828 6069954 6249867 5818936 5943423 6028938 6078909 6202157 6378072 6470450 6499110 6694434 6804777 6317700 6202159 5343527 5802557 6108788 6131162 6192131 6256734 6298153 6367009 6367012 6381324 6438690 6715073 6789194 6198053 6192474 5805911 5896321 6134597 6154747 4624480 6243811 6810389 4924515 4853697 5180259 5294121 5418854 5497421).pn. (713/200).ccls.	105

END OF SEARCH HISTORY